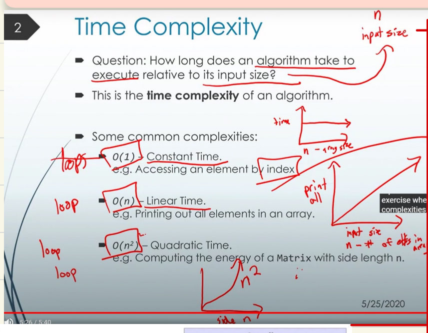
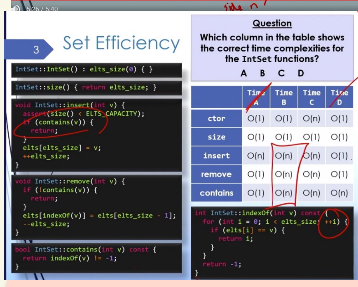
Video 1: time complexity



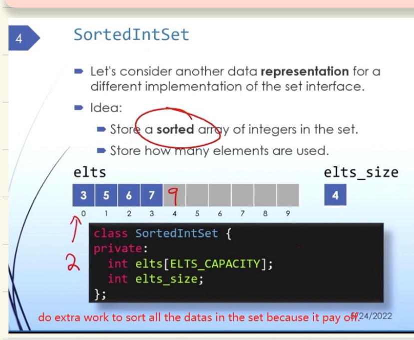
Time Complexity
Question: How long does an algorithm take to execute relative to its input size?
This is the time complexity of an algorithm.
Some common complexities:
O(1) – Constant Time.
e.g. Accessing an element by index.
O(n) – Linear Time.
e.g. Printing out all elements in an array.
O(n²) – Quadratic Time.
e.g. Computing the energy of a Matrix with side length n.
n input size
loops
loop
time
print all
input size n - # of elts in array
5/25/2020

① 3种 time complexities.



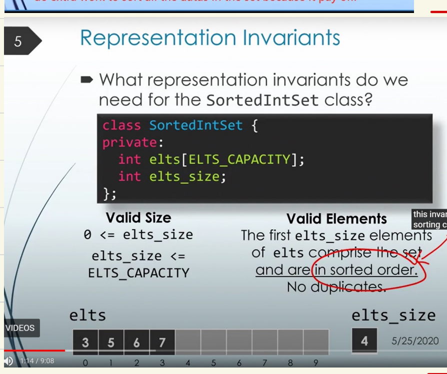
② Int Set 的 5个 member function 对应的 Complexity time.

Video 2: A sorted "IntSet"



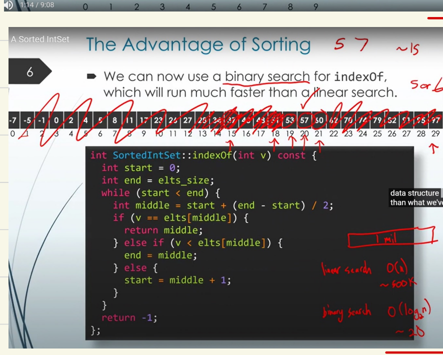
SortedIntSet
Let's consider another data representation for a different implementation of the set interface.
Idea:
Store a sorted array of integers in the set.
Store how many elements are used.
elts
elts_size
class SortedIntSet {
private:
int elts[ELTS_CAPACITY];
int elts_size;
};
do extra work to sort all the datas in the set because it pay off

① sorted array 中都需要哪些 private member datas (2).



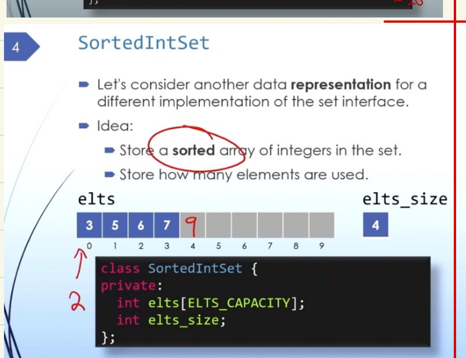
Representation Invariants
What representation invariants do we need for the SortedIntSet class?
class SortedIntSet {
private:
int elts[ELTS_CAPACITY];
int elts_size;
};
Valid Size
0 <= elts_size
elts_size <= ELTS_CAPACITY
Valid Elements
The first elts_size elements of elts comprise the set and are in sorted order.
No duplicates.
elts
elts_size
5/25/2020

①为了保证 representation 的 Validity, 我们须要怎样的 representation invariants (2)

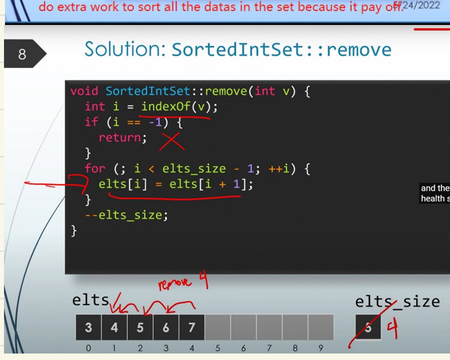


The Advantage of Sorting
We can now use a binary search for indexOf, which will run much faster than a linear search.
int SortedIntSet::indexOf(int v) const {
int start = 0;
int end = elts_size;
while (start < end) {
int middle = start + (end - start) / 2;
if (v == elts[middle]) {
return middle;
} else if (v < elts[middle]) {
end = middle;
} else {
start = middle + 1;
}
}
return -1;
};
linear search O(n)
binary search O(log n)

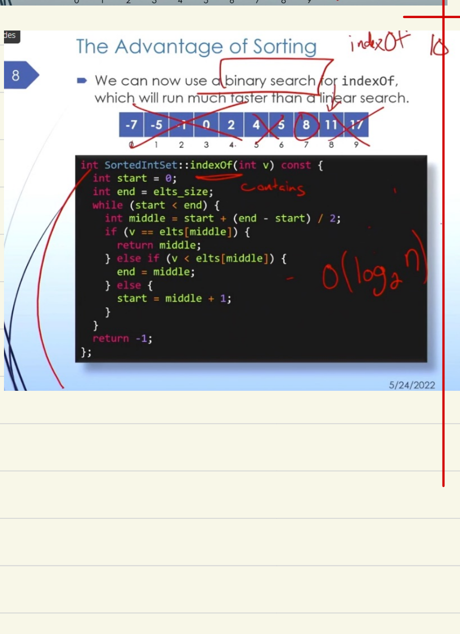
①如何使用 Binary search 提高运算速度.



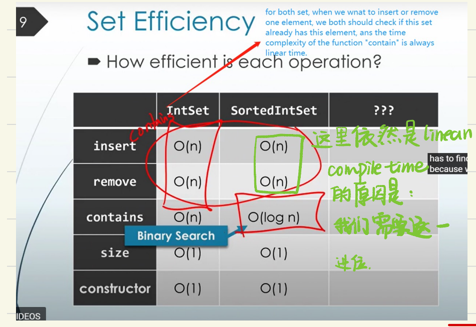
SortedIntSet
Let's consider another data representation for a different implementation of the set interface.
Idea:
Store a sorted array of integers in the set.
Store how many elements are used.
elts
elts_size
class SortedIntSet {
private:
int elts[ELTS_CAPACITY];
int elts_size;
};
do extra work to sort all the datas in the set because it pay off



①如何实现对于 sorted array "remove()" 的操作. (一种方式)



The Advantage of Sorting
We can now use a binary search for indexOf, which will run much faster than a linear search.
int SortedIntSet::indexOf(int v) const {
int start = 0;
int end = elts_size;
while (start < end) {
int middle = start + (end - start) / 2;
if (v == elts[middle]) {
return middle;
} else if (v < elts[middle]) {
end = middle;
} else {
start = middle + 1;
}
}
return -1;
};
contains
O(log₂ n)
5/24/2022



Set Efficiency
for both set, when we wnat to insert or remove one element, we both should check if this set already has this element, ans the time complexity of the function "contain" is always linear time.
How efficient is each operation?
IntSet
SortedIntSet
???
insert O(n) O(n)
remove O(n) O(n)
contains O(n) O(log n)
size O(1) O(1)
constructor O(1) O(1)
Binary Search
这里依然是 linear
compile time 的原因是: 我们需要逐一进位.

① Sorted set 的5个 member functions 的 compile time.

Exercise

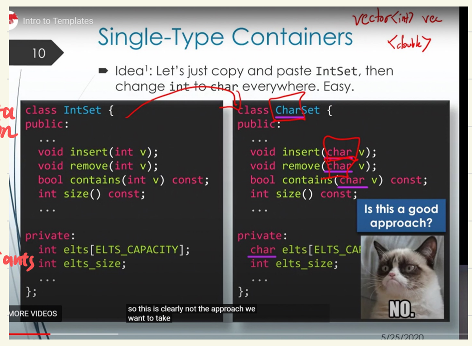
Implement the `insert()` member function for the `SortedIntSet` class. If the given value is not already in the set, it should be inserted into the `elts` array at the appropriate position to maintain the sorting invariant. Elements greater than the inserted value will need to be shifted to the right to create the space to insert the element. `elts_size` should also increase by 1. However, if the value is already in the array, `insert()` does nothing.

Starter code is available on Lobster (`L12.1_SortedIntSet`). The `main()` function provided includes testing code to verify your implementation.

Use the strategy described in the video/text above to shift over enough elements to make room for the inserted value, and don't forget (like I did in the video!) to call `contains()` as a helper function first to verify the number isn't already in the set (and just `return;` early if it is already there).

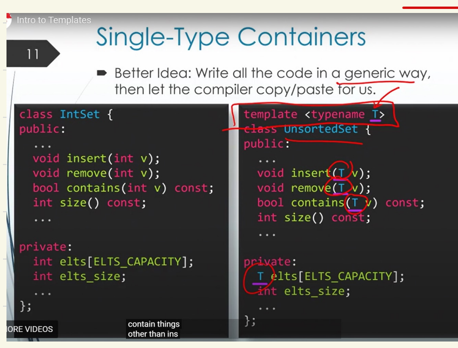
```
void SortedIntSet::insert(int v)  {
  // TODO: your code here
  if (contains(v)) { return; }

  // consider potential index i where we can put v
  int i = elts_size;
  for(; i > 0 && elts[i-1] > v; --i) {
    elts[i] = elts[i-1];
  }

  elts[i] = v;
  ++elts_size;
}
```

① insert 的实现. (从后往前)



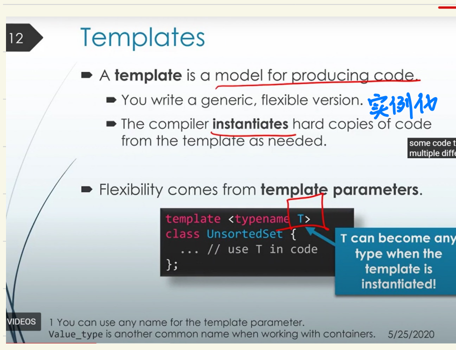
Single-Type Containers
Idea: Let's just copy and paste IntSet, then change int to char everywhere. Easy.
class IntSet {
public:
void insert(int v);
void remove(int v);
bool contains(int v) const;
int size() const;
private:
int elts[ELTS_CAPACITY];
int elts_size;
};
class CharSet {
public:
void insert(char v);
void remove(char v);
bool contains(char v) const;
int size() const;
private:
char elts[ELTS_CAPACITY];
int elts_size;
};
Is this a good approach?
NO.
vector<int> vec
<double>

①发现之前定义的 set class 只能针对一种数据类型.



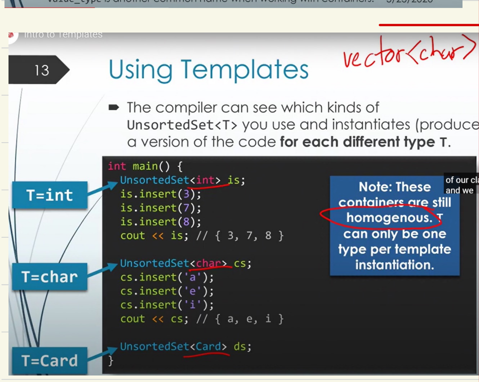
Single-Type Containers
Better Idea: Write all the code in a generic way, then let the compiler copy/paste for us.
class IntSet {
public:
void insert(int v);
void remove(int v);
bool contains(int v) const;
int size() const;
private:
int elts[ELTS_CAPACITY];
int elts_size;
};
template <typename T>
class UnsortedSet {
public:
void insert(T v);
void remove(T v);
bool contains(T v) const;
int size() const;
private:
T elts[ELTS_CAPACITY];
int elts_size;
};

①解决方法. 使用 template



Templates
A template is a model for producing code.
You write a generic, flexible version. 实例化
The compiler instantiates hard copies of code from the template as needed.
Flexibility comes from template parameters.
template <typename T>
class UnsortedSet {
... // use T in code
};
T can become any type when the template is instantiated!
1 You can use any name for the template parameter. Value_type is another common name when working with containers.
5/25/2020

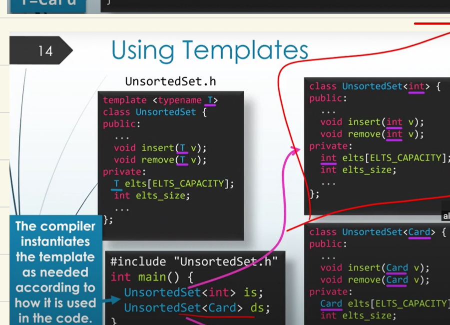
① Template 定义



Using Templates
The compiler can see which kinds of UnsortedSet<T> you use and instantiates (produces) a version of the code for each different type T.
int main() {
UnsortedSet<int> is;
is.insert(3);
is.insert(7);
is.insert(8);
cout << is; // { 3, 7, 8 }
UnsortedSet<char> cs;
cs.insert('a');
cs.insert('e');
cs.insert('i');
cout << cs; // { a, e, i }
UnsortedSet<Card> ds;
}
T=int
T=char
T=Card
Note: These containers are still homogenous. T can only be one type per template instantiation.
vector<char>

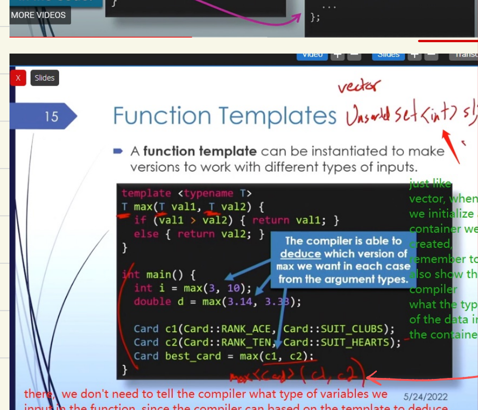
① template 的使用方式

② 一个 container 中, template 的 instantiation 必须一致.



Using Templates
UnsortedSet.h
template <typename T>
class UnsortedSet {
public:
void insert(T v);
void remove(T v);
private:
T elts[ELTS_CAPACITY];
int elts_size;
};
#include "UnsortedSet.h"
int main() {
UnsortedSet<int> is;
UnsortedSet<Card> ds;
}
The compiler instantiates the template as needed according to how it is used in the code.
class UnsortedSet<int> {
class UnsortedSet<Card> {

① 在 int main() 中使用 set class 时, 写在 "<>" 中表明其存储的 Variable type 进而 instantiate the template.

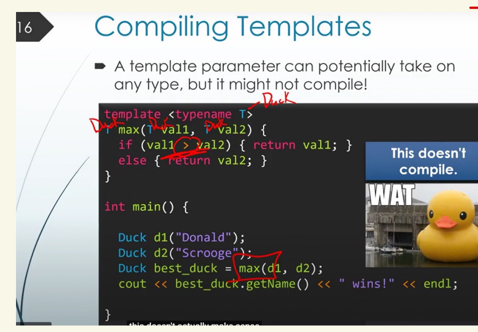


Function Templates
A function template can be instantiated to make versions to work with different types of inputs.
template <typename T>
T max(T val1, T val2) {
if (val1 > val2) { return val1; }
else { return val2; }
}
int main() {
int i = max(3, 10);
double d = max(3.14, 3.33);
Card c1(Card::RANK_ACE, Card::SUIT_CLUBS);
Card c2(Card::RANK_TEN, Card::SUIT_HEARTS);
Card best_card = max(c1, c2);
}
The compiler is able to deduce which version of max we want in each case from the argument types.
just like vector, when we initialize a container we created, remember to also show the compiler what the type of the data in the container
there, we don't need to tell the compiler what type of variables we input in the function, since the compiler can based on the template to deduce.
vector
UnsortedSet<int> s
max<Card>(c1, c2)
5/24/2022

The template is also called parametric polymorphism

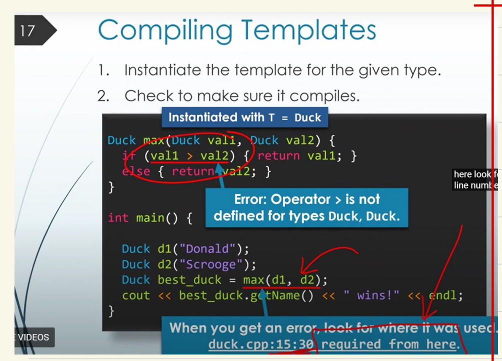
① 将 template 使用在 function 中

② 为什么我们不用告诉 compiler 我们输入 function 的 variable type

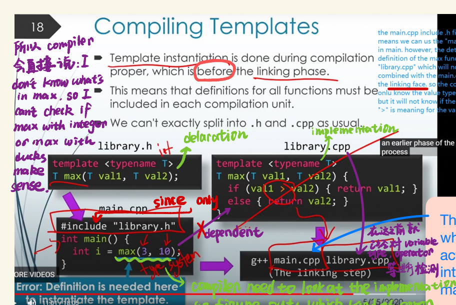


Compiling Templates
A template parameter can potentially take on any type, but it might not compile!
template <typename T>
T max(T val1, T val2) {
if (val1 > val2) { return val1; }
else { return val2; }
}
int main() {
Duck d1("Donald");
Duck d2("Scrooge");
Duck best_duck = max(d1, d2);
cout << best_duck.getName() << " wins!" << endl;
}
This doesn't compile.
WAT

① template 接收任何 variable type. 但不一定 make sense.



Compiling Templates
1. Instantiate the template for the given type.
2. Check to make sure it compiles.
Instantiated with T = Duck
Duck max(Duck val1, Duck val2) {
if (val1 > val2) { return val1; }
else { return val2; }
}
int main() {
Duck d1("Donald");
Duck d2("Scrooge");
Duck best_duck = max(d1, d2);
cout << best_duck.getName() << " wins!" << endl;
}
Error: Operator > is not defined for types Duck, Duck.
When you get an error, look for where it was used.
duck.cpp:15:30 required from here.



Compiling Templates
Template instantiation is done during compilation proper, which is before the linking phase.
This means that definitions for all functions must be included in each compilation unit.
We can't exactly split into .h and .cpp as usual.
library.h
template <typename T>
T max(T val1, T val2);
library.cpp
template <typename T>
T max(T val1, T val2) {
if (val1 > val2) { return val1; }
else { return val2; }
}
main.cpp
#include "library.h"
int main() {
int i = max(3, 10);
}
g++ main.cpp library.cpp
(The linking step)
Error: Definition is needed here to instantiate the template.
the main.cpp include .h file which means we can us the "max" function in main. however, the detail definition of the max function is in "library.cpp" which will not be combined with the main.cpp before the linking face, so the compiler will only know the value type of the "T", but it will not know if the compare ">" is meaning for the value type.
an earlier phase of the compilation process
因为 compiler 会直接说: I don't know what's in max, so I can't check if max with integer or max with ducks make sense.
declaration
implementation
since only dependent
compiler need to look at the implementation to figure out which kind of error type it should give

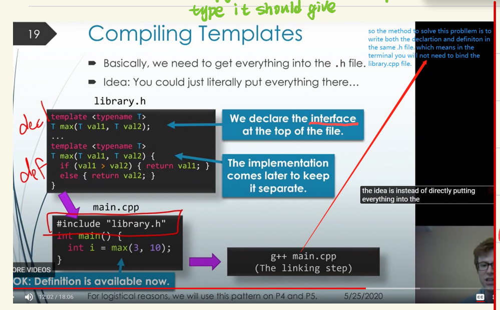
The compiler don't know what's in max, so i can't actually check if max with integers or max with ducks makes sense.

① template 实例化的时间.

② .cpp 和 .h 要合起来写.

③ implementation 和 declaration 之前都要加 template <typename T>

④ 分开写报错的原因.

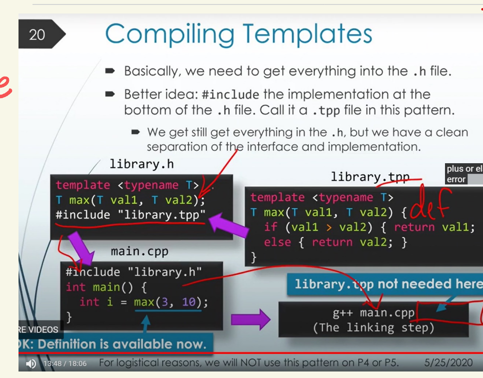


Compiling Templates
Basically, we need to get everything into the .h file.
Idea: You could just literally put everything there...
library.h
template <typename T>
T max(T val1, T val2);
...
template <typename T>
T max(T val1, T val2) {
if (val1 > val2) { return val1; }
else { return val2; }
}
We declare the interface at the top of the file.
The implementation comes later to keep it separate.
main.cpp
#include "library.h"
int main() {
int i = max(3, 10);
}
g++ main.cpp
(The linking step)
OK: Definition is available now.
so the method to solve this probllem is to write both the declartion and definiton in the same .h file, which means in the terminal you will not need to bind the library.cpp file.
dec
def

That's why when we use vector library, we only need to #include it in .cpp file and don't need to follow the library.cop file at the end of the main.cpp file when compile.

Both the prototype and the specific implementation are all in the same file.

①解决方法: 合并.

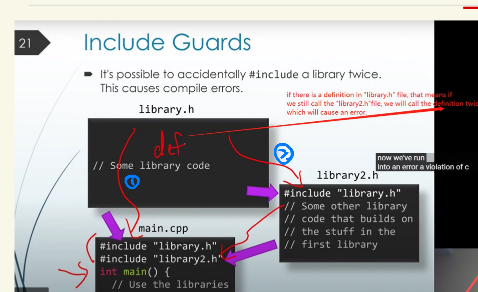


Compiling Templates
Basically, we need to get everything into the .h file.
Better idea: #include the implementation at the bottom of the .h file. Call it a .tpp file in this pattern.
We get still get everything in the .h, but we have a clean separation of the interface and implementation.
library.h
template <typename T>
T max(T val1, T val2);
#include "library.tpp"
library.tpp
template <typename T>
T max(T val1, T val2) {
if (val1 > val2) { return val1; }
else { return val2; }
}
main.cpp
#include "library.h"
int main() {
int i = max(3, 10);
}
library.tpp not needed here.
g++ main.cpp
(The linking step)
OK: Definition is available now.
For logistical reasons, we will NOT use this pattern on P4 or P5.
5/25/2020

① 用 tpp 代替 cpp.

② 引用 tpp 的位置

③ g++ 时的不同



Include Guards
It's possible to accidentally #include a library twice.
This causes compile errors.
library.h
// Some library code
library2.h
#include "library.h"
// Some other library
// code that builds on
// the stuff in the
// first library
main.cpp
#include "library.h"
#include "library2.h"
int main() {
// Use the libraries
}
if there is a definition in "library.h" file, that means if we still call the "library2.h" file, we will call the definition twice which will cause an error.
5/25/2020

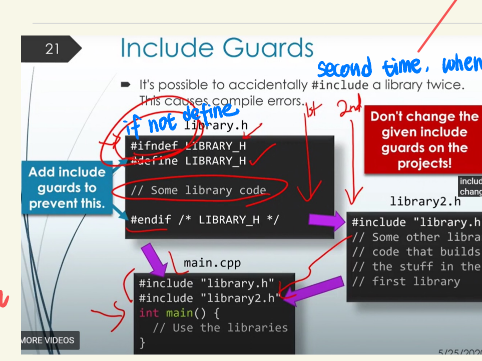
① 问题: 一个 library 可能被重复定义两遍.

② 解决:

ifndef

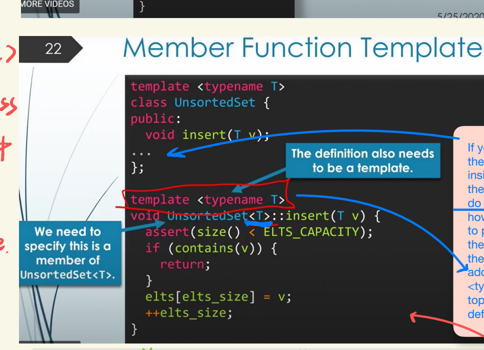
define

endif



Include Guards
It's possible to accidentally #include a library twice.
This causes compile errors.
library.h
#ifndef LIBRARY_H
#define LIBRARY_H
// Some library code
#endif /* LIBRARY_H */
Add include guards to prevent this.
Don't change the given include guards on the projects!
library2.h
#include "library.h"
// Some other library
// code that builds on
// the stuff in the
// first library
main.cpp
#include "library.h"
#include "library2.h"
int main() {
// Use the libraries
}
if not define
1st
2nd

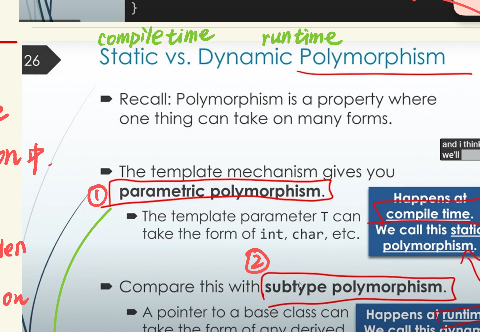
second time, when compiler found you already define the "library.h" it will skip every thing between "# define Library_H" and "#endif /* library_H */"



Member Function Templates
template <typename T>
class UnsortedSet {
public:
void insert(T v);
...
};
The definition also needs to be a template.
template <typename T>
void UnsortedSet<T>::insert(T v) {
assert(size() < ELTS_CAPACITY);
if (contains(v)) {
return;
}
elts[elts_size] = v;
++elts_size;
}
We need to specify this is a member of UnsortedSet<T>.

If you want to put all the implementation inside of the class, then you don't need to do anything special, however, if you want to put the definition of the function outside of the class, you should add another template <typename T> on the top of your. Function definition

① member function 定义在 class scope 里和外的区别.



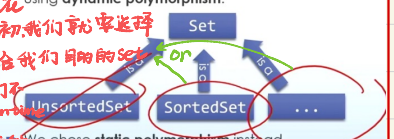
compile time
run time
Static vs. Dynamic Polymorphism
Recall: Polymorphism is a property where one thing can take on many forms.
The template mechanism gives you parametric polymorphism.
The template parameter T can take the form of int, char, etc.
Happens at compile time. We call this static polymorphism.
Compare this with subtype polymorphism.
A pointer to a base class can take the form of any derived class (i.e. use dynamic type).
Happens at runtime. We call this dynamic polymorphism.

① 两种多态指生的时和分类.

we I make a vector of int, I don't need to keep checking if it's still a vector of int. it's just always a vector of ints. I have the flexibility to have made it a vector of char. But once I chose in the compile time, it will fixed and never changed in the run time.

27 Static vs. Dynamic Polymorphism

- We could have implemented the two set ADTs using **dynamic polymorphism**.



- We chose **static polymorphism** instead.
  - Dynamic polymorphism requires member lookup at runtime, so there is an extra cost.
  - The type of set we're using doesn't change at runtime, so we don't need dynamic polymorphism.

对于使用者来说，在coding之初，我们就要选择一个最适合我们目的的set，所以我们不会有在runtime一个指向这种set，一会指向另一种set的需求

① 为什么不set中使用subtype polymorphism.

28 Using Sets

- With static polymorphism, if we decide to use SortedSet instead, there may be a lot of places where we need to change the type.

```
template <typename T>
void fillFromArray(UnsortedSet<T> &set, const T *arr,
                   int n);

int main() {
  UnsortedSet<int> set1;
  int arr1[4] = { 1, 2, 3, 2 };
  fillFromArray(set1, arr1, 4);

  UnsortedSet<char> set2;
  char arr2[3] = { 'a', 'b', 'a' };
  fillFromArray(set2, arr2, 3);
}
```

在这里我们使用了unsorted set，但如果我们后来想修改，就很麻烦。

① 问题：确认一种set，不好修改。

29 Type Aliases

- Instead, we can introduce a type alias with the `using` keyword.

```
template <typename T>
using Set = UnsortedSet<T>;

template <typename T>
void fillFromArray(Set<T> &set, const T *arr,
                   int n);

int main() {
  Set<int> set1;
  int arr1[4] = { 1, 2, 3, 2 };
  fillFromArray(set1, arr1, 4);

  Set<char> set2;
  char arr2[3] = { 'a', 'b', 'a' };
  fillFromArray(set2, arr2, 3);
}
```

Now this is the only place we need to change.

① 解决方法：使用type alias.